Construction guidée de requêtes analytiques sur des graphes RDF

Sébastien Ferré

Construction guidée de requêtes analytiques sur des graphes RDF

Sébastien Ferré*

*IRISA, Campus de Beaulieu, 35042 Rennes cedex, France,
ferre@irisa.fr,
http://people.irisa.fr/Sebastien.Ferre/

Résumé. Alors que de plus en plus de données sont disponibles sous forme de graphes RDF, la possibilité de faire des requêtes analytiques devient un enjeu important du Web des données. Les approches existantes nécessitent la modélisation de cubes de données au-dessus des graphes RDF. Nous proposons une approche qui permet de répondre à des requêtes analytiques directement sur des graphes RDF non modifiés, en exploitant les possibilités de calcul de SPARQL 1.1 (expressions, agrégations). Nous nous appuyons sur le patron de conception `N<A>F` pour concevoir une interface de construction guidée de requêtes qui est intuitive en cachant complètement SPARQL derrière une verbalization en langue naturelle, et qui est réactive en donnant des résultats intermédiaires et des suggestions à chaque pas. Nos évaluations montrent que notre approche couvre une large classe de cas d'utilisations et passe à l'échelle de grands graphes.

1 Introduction

La différence entre recherche (dans des données) et analyse de données peut être illustrée par la différence entre *Quels films ont été réalisés par Tim Burton ?* et *Combien de films sont produits chaque année et dans chaque pays ?*. L'analyse de données a été beaucoup étudiée dans les bases de données relationnelles avec les entrepôts de données et OLAP (Chaudhuri et Dayal, 1997) mais n'en est qu'au commencement dans le Web des données (Kämpgen et Harth, 2011; Hoefler et al., 2013; Colazzo et al., 2014; Höffner et al., 2016; Sherkhonov et al., 2017). La plupart de ces travaux adaptent l'approche OLAP aux graphes RDF. Typiquement, des administrateurs de données doivent d'abord extraire des cubes de données à partir de graphes RDF en spécifiant pour chaque cube ce que sont les observations, les dimensions et les mesures (Cyganiak et al., 2013). Les utilisateurs finaux peuvent alors utiliser les interfaces classiques de type OLAP pour visualiser les cubes et leur appliquer des transformations. Plus récemment, l'approche par questions-réponses (*Question Answering*) (Lopez et al., 2011) a été appliquée aux requêtes analytiques sur des cubes de données RDF (Unger et al., 2016; Höffner et al., 2016; Atzori et al., 2019) ; ainsi que l'approche de construction guidée de requêtes, par exemple en s'appuyant sur des patrons de requêtes pré-définis (Kovacic et al., 2018). Le principal inconvénient de l'usage de cubes de données est que les utilisateurs finaux n'ont plus d'accès direct au graphe original, et ne peuvent explorer que les cubes qui ont été définis

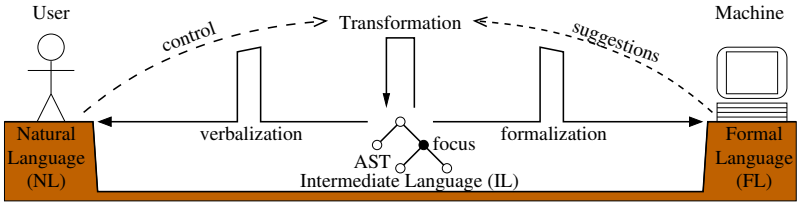

FIG. 1 – *Principe du patron de conception* N<A>F

par les administrateurs. Cet inconvénient est mitigé dans Colazzo et al. (2014) par un Schéma Analytique à partir duquel les utilisateurs peuvent dériver eux-mêmes de nombreux cubes. Cependant, il y a encore besoin d'un administrateur pour définir ce schéma.

Une conséquence du manque d'accès direct est une perte d'expressivité comparé à l'utilisation de SPARQL 1.1 (Kaminski et al., 2016). En effet, chaque vue OLAP peut être exprimée par une requête SPARQL où chaque ligne de résultat définit une cellule du cube avec ses coordonnées, alors que chaque cube de données ne permet qu'un ensemble limité de questions. Chaque nouvelle question peut nécessiter la définition d'un nouveau cube de données, ce qui nécessite généralement un administrateur. La contrepartie de l'expressivité de SPARQL est qu'il est bien plus difficile pour un utilisateur final d'écrire des requêtes SPARQL que d'intéragit via une interface OLAP. Un autre inconvénient de SPARQL est que les moteurs de requêtes ne sont pas optimisés pour les requêtes analytiques comme les moteurs OLAP. Cependant, ils sont déjà utilisables en pratique et leur optimisation dépasse le cadre de ce papier.

Dans ce papier, nous proposons une approche qui exploite les fonctionnalités de SPARQL 1.1 (expressions, agrégations) directement sur les graphes RDF non modifiés pour répondre à des requêtes analytiques. Nous nous appuyons sur le patron de conception N<A>F (Ferré, 2016) pour concevoir une interface de construction guidée de requêtes qui soit à la fois *intuitive* et *réactive*. Elle est intuitive en cachant complètement SPARQL derrière une verbalisation en langue naturelle des requêtes construites. Elle est aussi réactive en donnant des résultats intermédiaires et des suggestions de raffinement de la requête, à chaque pas d'interaction. Ainsi, notre interface utilisateur partage certains traits avec l'approche questions-réponses (verbalisation en langue naturelle) et avec OLAP (visualisation interactive).

Le papier est organisé comme suit. La section 2 présente succinctement le patron de conception N<A>F, qui sert de cadre formel à ce travail. La section 3 formalise la construction de requêtes analytiques dans le cadre N<A>F. La section 4 présente une évaluation sur le challenge QALD-6 et la section 5 présente une application aux données de Persée.

Le matériel en ligne associé à ce papier comprend une application web [1], les permaliens d'environ 70 requêtes analytiques et des captures vidéos (suivre `Learn > Examples` dans l'application web).

2 Le patron de conception N<A>F

La raison d'être du patron de conception N<A>F (Ferré, 2016) est de faire le pont entre les langues naturelles (NL) et les langages formels (FL, ici SPARQL). La figure 1 résume

1. http://www.irisa.fr/LIS/ferre/sparklis/

le principe de N<A>F. L'utilisateur est du côté NL et ne comprend pas FL. La machine, ici un point d'accès SPARQL, est du côté FL et ne comprend pas NL. L'élément central du pont est fait d'arbres de syntaxe abstraite (AST), appartenant à un langage intermédiaire (IL) entre NL et FL. IL est conçu pour rendre les traductions des AST vers NL (*verbalisation*) et FL (*formalisation*) aussi simple que possible. Il n'a pas besoin d'une syntaxe concrète, NL et FL jouent ce rôle respectivement pour l'utilisateur et la machine. Les AST sont des structures arborescentes où les nœuds représentent les sous-structures d'une requêtes. Chaque nœud $X = \mathbf{C}(X_1, \ldots, X_n)$ est caractérisé par un *constructeur* $\mathbf{C}$, le type du nœud, et un tuple de nœuds $X_1, \ldots, X_n$, les structures filles du nœud. N<A>F suit l'approche d'un constructeur de requêtes, où la structure qui est construite incrémentalement est précisément un AST. À la différence d'autres constructeurs de requêtes, la requête envoyée à la machine (FL) et la requête affichée à l'utilisateur (NL) peuvent fortement différer : IL joue un rôle de médiation. L'AST en construction est initialement la requête la plus simple et est incrémentalement affinée par application de *transformations*. Une transformation peut insérer ou supprimer une sous-structure de la requête. Chaque transformation s'applique au *focus* de la requête, où le focus est un nœud distingué de l'AST que l'utilisateur peut déplacer librement. Les transformations possibles sont suggérées par le système en fonction de la sémantique du langage de requêtes et des données, puis verbalisées en NL et enfin sélectionnées par l'utilisateur.

Nous illustrons N<A>F sur les motifs de graphe simples. L'AST

$$\mathbf{That}(\mathbf{A}(\texttt{:Film}), \mathbf{Has}(\texttt{:director}, \mathbf{Term}(\texttt{:Spielberg})))$$

est une imbrication de constructeurs inspirés des constructions syntaxique de la langue naturelle, avec des termes RDF comme sous-structures atomiques (e.g., la classe `:Film`, le terme `:Spielberg`). Les traductions en FL et NL sont réalisées par un parcours récursif de l'AST.

— SPARQL : `?x a :Film. ?x :director :Spielberg.`
— English : `'a film whose director is Spielberg'`
— French : `'un film dont le directeur est Spielberg'`

L'AST est construit via la séquence suivante de transformations. Après chaque transformation, nous donnons l'AST obtenu en soulignant la position du focus :

(0) AST initial : $\underline{\mathbf{Something}}$
(1) classe `:Film` : $\underline{\mathbf{A}(\texttt{:Film})}$
(2) propriété `:director` : $\mathbf{That}(\underline{\mathbf{A}(\texttt{:Film})}, \mathbf{Has}(\texttt{:director}, \underline{\mathbf{Something}}))$
(3) terme `:Spielberg` : $\mathbf{That}(\mathbf{A}(\texttt{:Film}), \mathbf{Has}(\texttt{:director}, \underline{\mathbf{Term}(\texttt{:Spielberg})}))$
(4) déplacement du focus : $\mathbf{That}(\underline{\mathbf{A}(\texttt{:Film})}, \mathbf{Has}(\texttt{:director}, \mathbf{Term}(\texttt{:Spielberg})))$

L'AST **Something** est traduit en SPARQL par une variable, donnant généralement accès aux valeurs d'une propriété (ex., le directeur des films en (2)). Sa verbalization de base est `'something'` mais se simplifie souvent comme en (2) où la verbalization de la requête est `'a film that has a director'`. Les transformations sont suggérées en fonction du focus et des résultats de la requête courante. Par exemple, en (2) seules les propriétés ayant pour sujet ou objet des films sont suggérées, et en (3) seuls les réalisateurs de films sont suggérés comme termes. À chaque étape, l'interface utilisateur montre : (a) la verbalisation de la requête courante avec le focus souligné, (b) les résultats de la requête SPARQL générée et (c) la liste des transformations suggérées.

3 Construction guidée de requêtes analytiques

La contribution de ce papier est de jeter un pont entre les utilisateurs et les possibilités de calcul de SPARQL 1.1 dans le but de permettre les requêtes analytiques directement sur les graphes RDF non modifiés. Nous réalisons ce pont comme une nouvelle instance du patron de conception N<A>F, c'est-à-dire en définissant de nouveaux AST avec leurs transformations, leur formalisation et leur verbalisation. Comparé à l'instance sur les motifs de graphes (voir la section 2), cette nouvelle instance introduit deux nouvelles sortes d'AST couvrant les calculs de SPARQL, en plus des motifs de graphe : les *tables* et les *expressions*. Les AST des requêtes analytiques sont de type table et contiennent des AST de type motif de graphe simple et/ou expression. La sémantique d'un AST P de type "motif de graphe simple" est un ensemble de valuations $M(P)$, où chaque valuation $\mu \in M$ est une fonction partielle des variables du motif vers des termes RDF. La sémantique d'un AST E de type "expression" est un terme RDF résultant d'un calcul $eval(E)$. La sémantique d'un AST T de type "table" est une structure tabulaire avec un ensemble de *colonnes* $C(T)$, qui sont des variables, et un ensemble de *lignes* $R(T)$, où chaque ligne $r \in R(T)$ est une fonction partielle des colonnes vers des termes RDF.

Dans les sous-sections suivantes, nous couvrons progressivement les possibilités de calcul de SPARQL en définissant de nouveaux constructeurs d'AST. Chaque sous-section présente un ou deux cas d'usage en guise de motivation, puis définit formellement le nouveau constructeur. Les cas d'usage sont basés sur un jeu de données réelles, MONDIAL (May, 1999), qui contient des données géographiques (ex., pays, villes, lacs) avec de nombreuses données numériques (ex., population, superficie).

Les constructeurs d'expressions sont définis d'un bloc dans la sous-section 3.2. Pour chaque constructeur d'un AST table T, nous définissons :

— $C(T)$: l'ensemble des colonnes,
— $R(T)$: l'ensemble des lignes,
— $sparql(T)$: la formalisation en SPARQL,
— $nl(T)$: la verbalisation en NL (ici, en anglais),
— un ensemble de transformations introduisant ce constructeur.

Une remarque importante est que la fonction $sparql()$ n'est pas toujours appliquée à l'AST table lui-même mais parfois à une variante de cet AST qui dépend du focus. Ces variantes sont expliquées ci-dessous quand nécessaire. La formalisation globale d'un AST table T est `SELECT * WHERE {` $sparql(T')$ `}` où T' est la variante de T dépendant du focus.

3.1 Tables primitives

Tous nos constructeurs de calcul définissent une table comme fonction d'une autre table. Nous avons donc besoin de constructeurs de tables primitives pour démarrer. Des candidats valables de tables primitives sont les tables des bases relationnelles, les cubes OLAP et les feuilles de calcul. Néanmoins, afin de permettre les requêtes analytiques directement sur des graphes RDF, nous proposons d'utiliser les motifs de graphe simples pour extraire des tables quelconques. En effet, les résultats d'une requête SPARQL (de type SELECT) prennent la forme d'une table. On peut ainsi réutiliser des travaux antérieurs sur la construction de motifs de graphes avec N<A>F, implémentée dans l'outil Sparklis (Ferré, 2017), afin de guider l'utilisateur dans la construction de ces tables primitives.

Définition 1 *Soit P un AST de motif de graphe simple. L'AST table $T = \mathbf{GetAnswers}(P)$ représente la table primitive où les lignes sont les valuations solutions de P.*

— $C(T) = Vars(sparql(P))$ — $sparql(T) = sparql(P)$
— $R(T) = M(P) = Result(sparql(P))$ — $nl(T) =$ `'give me` $nl(P)$`'`

Par exemple, si l'utilisateur veut construire la table primitive des pays avec leur population et leur superficie, 4 étapes sont suffisantes pour construire l'AST suivant :

$$T0 := \mathbf{GetAnswers}(\mathbf{That}(\mathbf{A}(\texttt{:Country}), \mathbf{And}(\mathbf{Has}(\texttt{:population}, \mathbf{Something}), \mathbf{Has}(\texttt{:area}, \mathbf{Something}))))$$

qui se traduit en SPARQL : `?x1 a :Country ; :population ?x2 ; :area ?x3.`
et en anglais : `'give me every country that has a population and that has an area'.`

3.2 Expressions dans BIND et FILTER

Cas d'usage (E) : *Donne-moi la densité de population pour chaque pays, à partir de la population et de la superficie*. Les expressions SPARQL (ex., `?x2 / ?x3`) sont utilisées dans deux sortes de contextes en SPARQL : *filtrage* (`FILTER`) and *définition* (`BIND`, `SELECT`, `GROUP BY`). Étant donné une table, un filtrage effectue une sélection sur l'ensemble des lignes sur la base d'une expression booléenne ; tandis qu'une définition ajoute une colonne et calcule ses valeurs pour chaque ligne avec l'expression. Dans les deux cas, l'expression ne peut faire référence qu'aux colonnes de la table. Les définitions permettent de dériver des informations qui ne sont pas explicitement représentées dans le graphe, ex. la densité de population d'une pays dans le cas d'usage (E).

Nous définissons d'abord les AST expression car ce sont des sous-structures des filtrages et des définitions. Nous les définissons de façon classique, comme une composition de constantes, de variables et de d'opérateurs/fonctions.

Définition 2 (expression) *Un AST expression E est composé de termes RDF (constantes), de colonnes de table (variables) et d'opérateurs/fonctions SPARQL. Nous ajoutons aussi le constructeur d'*expression indéfinie *?? pour permettre la construction incrémentale d'expressions. Une expression E est dite* définie *si elle ne contient pas de ??. On note $C(E)$ l'ensemble des colonnes utilisées par l'expression.*

La sémantique d'une expression E est notée $eval_r(E)$, où r est la ligne sur laquelle l'évaluation est effectuée. $eval_r(E)$ et $sparql(E)$ sont définies de façon évidente mais sont seulement définies quand E est une expression définie. Si le focus est sur une sous-expression E' alors seule cette sous-expression est évaluée et formalisée en SPARQL afin de montrer la valeur au focus. La verbalisation d'une expression résulte d'une imbrication des verbalisations de ses fonctions, opérateurs, colonnes et termes. Elle mélange des notations mathématiques et littérales selon lesquelles sont les plus claires : `'`E_1 `+` E_2`'` est clair pour tout le monde et moins verbeux que `'the addition of` E_1 `and` E_2`'`, tandis que `'`E_1 `or` E_2`'` est plus clair que `'`E_1 `||` E_2`'` pour de non informaticiens. La verbalisation des colonnes est dérivée des noms de classes/propriétés utilisées dans le motif de graphe qui les introduit en tant que variable :

ex., dans $sparql(T0)$, `'the population'` pour x_2, `'the area'` pour x_3. Les parties de l'expression qui ne sont pas sous le focus sont affichées en grisé pour indiquer qu'elles ne sont pas calculées.

Les transformations utilisées pour construire des AST expressions sont les suivantes : (a) insérer un terme RDF au focus, (b) insérer une colonne au focus, (c) appliquer un opérateur ou une fonction au focus. Quand un opérateur/fonction est appliqué, le focus est automatiquement déplacé à la prochaine sous-expression ?? s'il en existe une, typiquement un autre argument de la fonction si elle a plusieurs paramètres. Par exemple, la séquence de transformations et d'AST qui construit l'expression du cas (E) est la suivante :

(0) expression initiale : $E = \underline{??}$
(1) insérer la colonne x_2 (population) : $E = \underline{x_2}$
(2) appliquer l'opérateur / (division) : $E = x_2 \,/\, \underline{??}$
(3) insérer la colonne x_3 (area) : $E = \underline{x_2 \,/\, x_3}$

Il existe des contraintes sur les transformations applicables, de façon à éviter la construction d'expressions mal formées. Les contraintes sur les colonnes pouvant être insérées sont définies par les constructeurs de table les utilisant. Des contraintes de type sont aussi utilisées pour déterminer quels fonctions sont applicables en fonction du focus, des types de données des colonnes et des signatures des fonctions. Dans l'exemple précédent, à l'étape (2), seuls des fonctions numériques peuvent être appliquées car la colonne x_2 contient des entiers ; et à l'étape (3), seules les colonnes numériques peuvent être insérées.

Nous définissons maintenant les deux constructeurs de tables qui utilisent une expression.

Définition 3 (filtrage) *Soit T_1 un AST table, et E un AST d'expression booléenne tel que $C(E) \subseteq C(T_1)$. L'AST table $T = \mathbf{SelectRows}(T_1, E)$ représente le filtrage des lignes de T_1 qui vérifient E.*

— $C(T) = C(T_1)$
— $R(T) = \begin{cases} \{r \mid r \in R(T_1), eval_r(E) = true\} & \textit{si } E \textit{ est définie}, \\ R(T_1) & \textit{sinon} \end{cases}$
— $sparql(T) = \begin{cases} sparql(T_1)\ \texttt{FILTER (}\ sparql(E)\ \texttt{)} & \textit{si } E \textit{ est définie} \\ sparql(T_1) & \textit{sinon} \end{cases}$
— $nl(T) =$ `'`$nl(T_1)$ `where` $nl(E)$`'`

Définition 4 (définition) *Soit T_1 un AST table, $x \notin C(T_1)$ une variable fraîche, et E un AST expression tel que $C(E) \subseteq C(T_1)$. L'AST table $T = \mathbf{AddColumn}(T_1, x, E)$ représente l'ajout d'une colonne x à T_1, et la définition de cette nouvelle colonne par E.*

— $C(T) = C(T_1) \cup \{x\}$
— $R(T) = \begin{cases} \{r \cup \{x \mapsto eval_r(E)\} \mid r \in R(T_1)\} & \textit{si } E \textit{ est définie}, \\ R(T_1) & \textit{sinon } (x \textit{ est non défini}) \end{cases}$
— $sparql(T) = \begin{cases} sparql(T_1)\ \texttt{BIND (}\ sparql(E)\ \texttt{AS ?}x\ \texttt{)} & \textit{si } E \textit{ est définie}, \\ sparql(T_1) & \textit{sinon} \end{cases}$
— $nl(T) = \begin{cases} \texttt{`}nl(T_1)\ \texttt{and give me}\ name(x)\ \texttt{=}\ nl(E)\texttt{'} & \textit{si } name(x) \textit{ est défini}, \\ \texttt{`}nl(T_1)\ \texttt{and give me}\ nl(E)\texttt{'} & \textit{sinon} \end{cases}$

Dans les deux constructeurs, les colonnes utilisables par l'expressions sont celles de T_1. Si le focus est sur une sous-expression E', alors la fonction $sparql()$ s'applique respectivement à $T' = \mathbf{SelectRows}(T_1, E')$ et $T' = \mathbf{AddColumn}(T_1, x, E')$, ignorant ainsi le reste de l'expression dans le calcul. Il suffit de déplacer le focus à la racine de l'AST expression ou au-dessus afin de récupérer le calcul complet.

Les filtrages et définitions sont introduits dans l'AST en mettant le focus sur une colonne et en lui appliquant une fonction ou un opérateur, ce qui initie la construction d'une expression. Le choix entre **SelectRows** et **AddColumn** est basé sur le type de l'expression entière selon l'hypothèse que les expressions booléennes sont généralement utilisées pour filtrer les lignes d'une table. Cependant, une autre transformation permet de forcer le choix de **AddColumn** pour obtenir des colonnes de valeurs booléennes. Enfin, une autre transformation permet d'attribuer un nom utilisateur $name(x)$ à la nouvelle colonne introduite par une définition, lequel est ensuite utilisé dans la verbalisation. Par exemple, en partant de la table primitive $T0$ définie dans la section précédente, le cas (E) peut être construit via la séquence suivante de transformations d'AST :

(0-4)	...	: $T = T0$ (voir la section 3.1)
(5)	focus sur x_2 (population)	
(6)	appliquer l'opérateur /	: $T = \mathbf{AddColumn}(T0, x_4, x_2 \ / \ \underline{??})$
(7)	insérer la colonne x_3 (area)	: $T = \mathbf{AddColumn}(T0, x_4, \underline{x_2 \ / \ x_3})$
(8)	$name(x_4) :=$ `'population density'`	

L'AST table résultant peut être traduit en SPARQL :

```
?x1 a :Country ; :population ?x2 ; :area ?x3. BIND (?x2/?x3 AS ?x4)
```

et en anglais :

```
'give me every country that has a population and that has an area,
 and give me the population density = the population / the area'.
```

3.3 Agrégations

Une *agrégation de base* est l'application d'un opérateur d'agrégation (ex., somme, moyenne) sur un ensemble d'entités ou valeurs, résultant en une unique valeur. Cas d'usage (A1) : *Combien de pays existe-t-il en Europe ?*. Une *agrégation simple* consiste à faire des groupes à partir d'un ensemble de valeurs selon un ou plusieurs critères, puis à appliquer un opérateur d'agrégation sur chaque groupe de valeurs. Une *agrégation multiple* étend l'agrégation simple en produisant plusieurs valeurs agrégées par groupe. Cas d'usage (A2) : *Donne-moi les moyennes de la population et de la superficie des pays, pour chaque continent*. Une agrégation correspond à une vue OLAP où les critères de regroupement sont les *dimensions* du cube et où les valeurs agrégées en sont les *mesures*. En SPARQL, les agrégations reposent sur l'usage d'opérateurs d'agrégation dans la clause `SELECT` et sur les clauses `GROUP BY`. Nous introduisons maintenant un nouveau constructeur de table qui couvre tous les types d'agrégations ci-dessus.

Définition 5 (agrégations) *Soit T_1 un AST table, $X \subseteq C(T_1)$ un ensemble de colonnes (possiblement vide), et $G = \{(g_j, y_j, z_j)\}_{j \in 1..m}$ un ensemble de triplets de la forme (opérateur d'agrégation, colonne, colonne) tels que pour tout $j \in 1..m$, $y_j \in C(T_1) \setminus X$ and $z_j \notin C(T_1)$. L'AST table $T = Aggregate(T_1, X, G)$ représente la table obtenue en regroupant les lignes*

de T_1 selon les colonnes X et, pour chaque $(g_j, y_j, z_j) \in G$, en définissant la nouvelle colonne z_j par l'application de g_j au multi-ensemble des valeurs de y_j dans chaque groupe.

- $C(T) = X \cup \{z_j\}_{j \in 1..m}$
- $R(T) = \{r_X \cup \{z_j \mapsto g_j(V_j)\}_{j \in 1..m} \mid r_X \in \pi_X R(T_1)$, *pour chaque* $(g_j, y_j, z_j) \in G$, $V_j = \{\{r(y_j) \mid r \in R(T_1), \pi_X r = r_X\}\}\ \}$
- $sparql(T) =$ `{ SELECT ?`$x_1 \ldots$`?`x_n `(`g_1`(?`y_1`) AS ?`z_1`) ... (`g_m`(?`y_m`) AS ?`z_m`) WHERE {` $sparql(T_1)$ `} GROUP BY ?`$x_1 \ldots$`?`x_n `}`
- $nl(T) =$ `'`$nl(T_1)$ `and [for ...each` $nl(x_i)$`, ...] give me ...`$nl(z_j)$`...'`

Dans $C(T)$, les colonnes y disparaissent et sont remplacées par les colonnes agrégées z. Dans la définition de $R(T)$, la notation $\{\{...\}\}$ représente un multi-ensemble. En effet, la distinction avec les ensembles est importante pour les opérateurs d'agrégation `SUM` et `AVG`. Comme les autres constructeurs de tables génèrent des motifs de graphe SPARQL, pas des requêtes, nous utilisons dans la formalisation des agrégations une sous-requête SPARQL (requête SPARQL entre accolades). Cela permet l'imbrication arbitraires des différents constructeurs de tables. Si le focus est dans T_1, alors les fonctions $R()$ et $sparql()$ sont appliquées à T_1 au lieu de T, ignorant ainsi l'agrégation. Cela permet d'accéder temporairement aux colonnes cachées par l'agrégation. La verbalisation de chaque colonne agrégée z_j est la verbalisation de $g_j(y_j)$: ex., `'the number of` $nl(y_j)$`'`, `'the average` $nl(y_j)$`'`. Les crochets autour de `'for each...'` indiquent une partie optionnelle, vide dans le case où $X = \emptyset$.

De façon similaire aux filtrages et définitions de colonnes, une agrégation est introduite dans un AST en déplaçant le focus sur une colonne et en y appliquant un opérateur d'agrégation. Ensuite, d'autres colonnes peuvent être sélectionnées, soit comme critère de regroupement, soit comme autre colonne à agréger. Les contraintes de type sont aussi utilisées ici pour déterminer quels opérateurs peuvent être appliqués à une colonne. La séquence de transformations qui répond au cas (A2) à partir de la table primitive $T0$ est la suivante :

(0-4) ... : $T = T0$ (voir la section 3.1)
(5) propriété `:continent` : $T = T1 = \mathbf{GetAnswers}(\ldots, \mathbf{Has}(\texttt{:continent}, \ldots)$
(6) déplacer le focus sur x_2 (population)
(7) appliquer agrégation `AVG` : $T = \mathbf{Aggregate}(T1, \{\}, \{(\texttt{AVG}, x_2, x_5)\})$
(8) grouper par x_4 (continent) : $T = \mathbf{Aggregate}(T1, \{x_4\}, \{(\texttt{AVG}, x_2, x_5)\})$
(9) appliquer `AVG` sur x_3 (area) : $T = \mathbf{Aggregate}(T1, \{x_4\}, \{(\texttt{AVG}, x_2, x_5), (\texttt{AVG}, x_3, x_6)\})$

Différentes séquences sont possibles, les éléments pouvant être introduits dans presque n'importe quel ordre. Il est également possible de supprimer des éléments pour faire évoluer la requête sans recommencer de zéro. Par exemple, dans le cas (A2), il est possible de construire d'abord la requête sans la colonne x_4 (continent), et sans grouper par x_4. Cela calcule les moyennes des populations et superficies de tous les pays. Ensuite, on peut remettre le focus sur x_1 (country), insérer la propriété `:continent` pour ajouter la colonne x_4 dans la table avant agrégation, et pour finir mettre le focus sur l'agrégation et grouper par continent. L'AST résultant pour le cas (A2) se traduit en SPARQL par :

```
{ SELECT ?x4 (AVG(?x2) AS ?x5) (AVG(?x3) AS ?x6)
  WHERE { ?x1 a :Country. ?x1 :population ?x2.
          ?x1 :area ?x3. ?x1 :continent ?x4. }
  GROUP BY ?x4 }
```

et en anglais par :

```
‘ give me every country
      that has a population
      and that has an area
      and that has a continent
and for each continent,
      give me the average population and the average area ’
```

3.4 Combinaisons de constructeurs de tables

Le véritable atout de notre approche tient en la possibilité de combiner les différents constructeurs de tables (**SelectRows**, **AddColumn**, **Aggregate**) et donc les différents type de calculs correspondants. Nous illustrons cela avec trois nouveaux cas d'utilisation.

Agrégation de définitions. Cas d'usage (C1) : *Quel est le PIB par habitant moyen par continent ?* Ce cas nécessite de calculer le PIB par habitant via l'expresion (PIB total $\times 10^6$/ population) pour chaque pays (définition), puis de faire la moyenne des PIB par habitant pour chaque continent (agrégation simple). La requête peut être construite en 13 étapes, produisant une imbrication de 3 constructeurs de tables.

Comparaison d'agrégations. Cas d'usage (C2) : *Quels continents ont un PIB agricole moyen supérieur à leur PIB tertiaire moyen ?* Ce case nécessite de calculer deux agrégations pour le même critère 'continent' (agrégation multiple), puis d'exprimer une inégalité entre les deux (filtrage). La requête peut être construite en 9 étapes, produisant une imbrication de 3 constructeurs de tables.

Agrégations imbriquées. Cas d'usage (C3) : *Donne-moi, pour chaque nombre d'îles possible dans un archipel, combien d'archipels ont ce nombre d'îles.* Ce cas nécessite premièrement de calculer le nombre d'îles par archipel (agrégation simple), et deuxièmement de calculer pour chacun de ces nombres d'îles le nombre d'archipels ayant ce nombre d'îles (agrégation simple). La requête peut être construite en 6 étapes, avec 3 constructeurs de table imbriqués.

3.5 Implémentation

Nous avons complètement implémenté notre approche dans l'outil Sparklis. Sa version de base couvrait les motifs de graphes et fournissait donc tout ce qu'il faut pour construire nos tables primitives. Grâce à la généricité de `N<A>F`, aucun changement n'est nécessaire dans l'interface utilisateur. L'impact de notre approche apparaît seulement à l'utilisateur au travers d'un langage de requête plus riche et de suggestions supplémentaires. Cela facilite la transition vers la nouvelle version. En revanche, dans l'implémentation, un remaniement important du langage intermédiaire a été nécessaire avec l'introduction des AST de tables et d'expressions en plus des AST de motifs de graphe. De nouveaux composants ont été introduits, par exemple pour l'inférence et la vérification de types dans le calcul des suggestions.

4 Évaluation

Le challenge QALD-6 (*Questions Answering over Linked Data*) a introduit une nouvelle tâche intitulée “Questions-réponses statistiques sur des cubes de données RDF” (Unger et al.,

2016). Le jeu de données contient environ 4 millions de transactions sur des dépenses gouvernementales dans le monde, organisée en 50 cubes de données. Il contient environ 16M triplets au total. Bien que le jeu de données soit représenté sous forme d'un ensemble de cubes, nous avons répondu aux questions du challenge en construisant des tables primitives à partir de motifs de graphes, comme nous le ferions pour un quelconque jeu de données RDF. Nous avons officiellement participé au challenge en soumettant les réponses obtenues en construisant des requêtes dans notre implémentation.

Expressivité. Nous évaluons l'expressivité de notre approche en mesurant la couverture des questions de QALD-6. Sur les 150 questions (entraînement+test), 148 questions sont toutes des agrégations basiques ou simples et sont donc couvertes par notre approche ; et 2 questions (training-Q23 et test-Q23) sont des comparaisons de deux agrégations et ne sont pas couvertes par notre approche. La raison est que les deux agrégations utilisent des motifs de graphe disjoints, alors que notre approche est limitée à un seul motif de graphe connexe.

Correction. Dans le challenge, nous sommes donc parvenus à répondre à 49/50 questions tests. Sur les 49 réponses soumises, 47 étaient correctes, d'où un taux de succès de 94% (mesure officielle : $F_1 = 0.95$). Les deux erreurs proviennent d'une ambiguité dans les questions Q35 et Q42, qui admettent plusieurs réponses aussi plausibles l'une que l'autre du fait que plusieurs URI ont la même étiquette. Bien que notre approche ne soit pas directement comparable aux autres participants qui ont utilisé l'approche questions-réponses en langue naturelle, il est utile de mentionner que leurs taux de succès étaient de 50% pour QA[3] ($F_1 = 0.53$), et de 38% pour CubeQA ($F_1 = 0.44$). Même si notre approche n'est pas aussi intuitive que les questions-réponses, elle fait la démonstration de sa pertinence en terme de taux de correction, tout en étant bien plus intuitive qu'écrire des requêtes en SPARQL. De plus, les autres approches nécessitent une représentation sous forme de cubes de données et ne sont donc pas directement applicables aux graphes RDF classiques.

Réactivité. Nous évaluons la réactivité du système en mesurant le temps nécessaire pour construire les requêtes dans notre implémentation par quelqu'un en maîtrisant bien l'interface (l'auteur de ce papier). Nos mesures de temps incluent les interactions utilisateur et sont donc une borne supérieure du temps d'éxécution par le système. Le temps de construction des requêtes va de 31s à 6min20s, et la moitié des requêtes sont construites en moins de 1min30s (temps médian). La plupart des questions de QALD-6 nécessitent 5-10 étapes. Cela démontre que notre implémentation est suffisamment réactive pour satisfaire de véritables besoins d'information sur de grands jeux de données. Cependant, cela ne prouve rien concernant le temps nécessaire à de véritables utilisateurs pour construire les requêtes (utilisabilité).

5 Application

Persée [2] est une organisation (UMS CNRS) qui fournit un accès libre à une collection de plus de 600.000 publications scientifiques, notamment dans le domaine des humanités et sciences sociales. Il maintient un point d'accès SPARQL qui donne accès à leurs méta-données [3]. L'outil Sparklis a été officiellement adopté par Persée en février 2017 comme outil de recherche dans ces méta-données RDF. Nous avons consulté des membres de Persée pour savoir quels types de requêtes les intéressaient et nous avons été surpris de constater que parmi

2. `http://www.persee.fr/`
3. `http://data.persee.fr/`

celles-ci beaucoup étaient des requêtes analytiques. Elles couvrent d'ailleurs toutes les catégories de requêtes présentées dans ce papier. Nous donnons un échantillon représentatif de ces requêtes analytiques, avant de les commenter.

- Q1 *Combien de documents a une personne donnée (ex., Pierre Bourdieu) co-écrit avec chacun de ses co-auteurs ?*
- Q2 *Pour chaque auteur, obtenir le nombre et la liste des titres de ses documents.*
- Q3 *Comment a évolué le nombre moyen d'auteurs par document à travers le temps ?*
- Q4 *Y a-t-il des personnes dont la date de décès est antérieure à la date de naissance, révélant ainsi des erreurs dans les données ?*
- Q5 *Trier les auteurs par durée décroissante de leur période d'activité, laquelle est calculée comme la différence entre les dates maximale et minimale de publication de leurs documents.*
- Q6 *Pour chaque année de publication, obtenir le nombre d'articles publiés cette année, et dont le titre contient un terme donné (ex., "rural"), afin d'étudier l'évolution de ce terme dans le temps.*

Q1 et Q2 sont des agrégations, multiple dans le cas de Q2 ("la liste des" se traduit en SPARQL par `GROUP_CONCAT`). Q3 est une agrégation imbriquée, comptant tout d'abord les auteurs par document, puis faisant la moyenne de ces nombres d'auteurs par année de publication. Q4 implique un filtrage dont l'expression combine deux propriétés des personnes (dates de naissance et de décès). Q5 et Q6 sont des combinaisons complexes d'agrégations et de définitions. Dans Q5, on a d'abord deux agrégations, une pour le minimum et l'autre pour le maximum, puis une définition pour la différence entre le maximum et le minimum, et enfin un tri. Dans Q6, on a tout d'abord un filtrage par le terme choisi, puis une agrégation par année. Toutes les questions reçues ont pu être résolues dans notre implémentation. Cela conforte le niveau d'expressivité de notre approche.

Il est important de noter que le jeu de données de Persée n'est pas du tout organisé en cubes de données, et ne contient même pas de données numériques à part les dates. La propriété centrale est celle reliant les documents aux auteurs, qui est une relation n-n. Comme les questions ci-dessus le montrent, les documents et les auteurs servent autant de dimension que de mesure. Ces cas d'usage réel soutiennent donc notre approche directe des requêtes analytiques en RDF.

6 Conclusion

Nous avons montré que la puissance de SPARQL 1.1 pouvait être exploitée pour faire des requêtes analytiques sur des graphes RDF non modifiés au travers d'une interface de construction guidée de requêtes à la fois intuitive et réactive. L'approche a été implémentée dans Sparklis, évaluée sur le challenge QALD-6 et appliquée dans le graphe de connaissances de Persée.

Références

Atzori, M., G. M. Mazzeo, et C. Zaniolo (2019). Qa3 : A natural language approach to question answering over rdf data cubes. *Semantic Web 10*(3), 587–604.

Chaudhuri, S. et U. Dayal (1997). An overview of data warehousing and OLAP technology. *ACM Sigmod record 26*(1), 65–74.

Colazzo, D., F. Goasdoué, I. Manolescu, et A. Roatiş (2014). RDF analytics : lenses over semantic graphs. In *Int. Conf. World Wide Web*, pp. 467–478. ACM.

Cyganiak, R., D. Reynolds, et J. Tennison (2013). The RDF data cube vocabulary.

Ferré, S. (2016). Bridging the gap between formal languages and natural languages with zippers. In H. Sack et al. (Eds.), *Extended Semantic Web Conf. (ESWC)*, pp. 269–284. Springer.

Ferré, S. (2017). Sparklis : An expressive query builder for SPARQL endpoints with guidance in natural language. *Semantic Web : Interoperability, Usability, Applicability 8*(3), 405–418.

Hoefler, P., M. Granitzer, V. Sabol, et S. Lindstaedt (2013). Linked data query wizard : A tabular interface for the semantic web. In *The Semantic Web : ESWC 2013 Satellite Events*, pp. 173–177. Springer.

Höffner, K., J. Lehmann, et R. Usbeck (2016). CubeQA - question answering on RDF data cubes. In *Int. Semantic Web Conf.*, pp. 325–340. Springer.

Kaminski, M., E. V. Kostylev, et B. Cuenca Grau (2016). Semantics and expressive power of subqueries and aggregates in SPARQL 1.1. In *Int. Conf. World Wide Web*, pp. 227–238. ACM.

Kämpgen, B. et A. Harth (2011). Transforming statistical linked data for use in OLAP systems. In *Int. Conf. Semantic systems*, pp. 33–40. ACM.

Kovacic, I., C. G. Schuetz, S. Schausberger, R. Sumereder, et M. Schrefl (2018). Guided query composition with semantic OLAP patterns. In *EDBT/ICDT Workshops*, pp. 67–74.

Lopez, V., V. S. Uren, M. Sabou, et E. Motta (2011). Is question answering fit for the semantic web ? : A survey. *Semantic Web 2*(2), 125–155.

May, W. (1999). Information extraction and integration with FLORID : The MONDIAL case study. Technical Report 131, Universität Freiburg, Institut für Informatik. Available from `http://dbis.informatik.uni-goettingen.de/Mondial`.

Sherkhonov, E., B. C. Grau, E. Kharlamov, et E. V. Kostylev (2017). Semantic faceted search with aggregation and recursion. In C. d'Amato et al. (Eds.), *Int. Semantic Web Conf. (ISWC)*, LNCS 10587, pp. 594–610. Springer.

Unger, C., A.-C. N. Ngomo, et E. Cabrio (2016). 6th open challenge on question answering over linked data (QALD-6). In H. Sack et al. (Eds.), *Semantic Web Evaluation Challenge*, pp. 171–177. Springer.

Summary

As more and more data are available as RDF graphs, the availability of tools for analytical queries beyond semantic search becomes a key issue of the Semantic Web. Previous work require the modelling of data cubes on top of RDF graphs. We propose an approach that directly answers analytical queries on unmodified RDF graphs by exploiting the computation features of SPARQL 1.1 (aggregations, expressions). We rely on the `N<A>F` design pattern to design a query builder user interface that is user-friendly by completely hiding SPARQL behind a verbalization in natural language; and responsive by giving intermediate results and suggestions at each step. Our evaluations show that our approach covers a large range of use cases, and scales well on large datasets.